Politechnika Warszawska

WYDZIAŁ ELEKTRYCZNY

Instytut Elektrotechniki Teoretycznej i Systemów Informacyjno-Pomiarowych

Praca dyplomowa magisterska

na kierunku Informatyka Stosowana

w specjalności Inżynieria Oprogramowania

Porównanie efektywności wybranych narzędzi służących do serwowania danych

inż. Jan Łukomski

numer albumu 291089

promotor

prof. dr hab. inż. Remigiusz Rak

WARSZAWA 2024

Porównanie efektywności wybranych narzędzi służących do serwowania danych

Streszczenie

TODO na całą stronę

Słowa kluczowe: A, B, C

Comparison of the effectiveness of selected data serving tools

Abstract

TODO

Keywords: X, Y, Z

Spis treści

Rozdział 1

Wstęp

Wybór technologii w ramach projektu jest decyzją strategiczną, którą nie jest łatwo zmienić na późniejszym etapie. W celu utrzymywania produktu warto czasami podjąć trudną decyzję o zmodernizowaniu stosu techonologiczengo. Technologię zawsze dobiera się na podstawie wielu okoliczności, które uzależnione są od czasem nieoczywistych warunków.

1.1 Cel i zakres pracy

Praca ta dokonuje analizy porównawczej trzech wybranych frameworków serwujących dane. Pod uwagę został wzięty aspekt czasu odpowiedzi przy róznych warunkach. Badanie ma na celu wzkazać zalety oraz wady każdego z porównywanych narzędzi.

Rozdział 2

Przegląd literaturowy

Rozdział 3

Materiał badawczy

3.1 Django

Django to framework do tworzenia aplikacji internetowych [3]. Został wydany w 2005 roku. Nazwa pochodzi od gitarzysty Django Reinhardta. Jest to narzędzie, które nadaje się do szybkiego tworzenia zaawansowanych aplikacji internetowych, zachowując jednocześnie wysoką jakość i bezpieczeństwo kodu. Jego popularność wynika z wielu czynników, w tym szybkości wdrażania projektów, bogatej palety wbudowanych funkcji oraz zabezpieczeń.

Jedną z głównych zalet Django jest jego zdolność do szybkiego rozwoju aplikacji od pomysłu do wdrożenia. Framework ten oferuje wiele narzędzi i gotowych rozwiązań, które ułatwiają proces tworzenia stron internetowych, od autentykacji użytkowników po obsługę treści czy generowanie map strony. Dzięki temu programiści mogą skupić się na kształtowaniu logiki aplikacji. Posiada on wbudowane moduły, które pozwalają na szybki start projektu. Są to mechanizmy obrony przed najczęstszymi atakami, takimi jak wstrzykiwanie SQL czy ataki typu cross-site scripting, czy system autentykacji użytkowników, zarządzanie kontami użytkowników oraz hasłami.

Django pozwala aplikacjom na elastyczne dostosowywanie się do zmieniających się wymagań i wzrostu liczby użytkowników. Może być stosowany zarówno w małych projektach, jak i w dużych systemach obsługujących ogromne ruchy internetowe, co czyni go uniwersalnym narzędziem dla różnorodnych zastosowań.

Kod źródłowy frameworka Django jest udostępniony publicznie na zasadach otwartego oprogramowania.

Strony takie jak Pinterest, czy Instagram korzystają z frameworku Django do swojego działania.

3.2 .NET

.NET Framework, rozwijany przez firmę Microsoft od 2002 roku, stanowi kompleksową platformę programistyczną, która umożliwia tworzenie różnorodnych aplikacji komputerowych [1]. Jest to zintegrowane środowisko programistyczne zawierające narzędzia, biblioteki oraz środowisko wyko-

nawcze, które ułatwiają proces tworzenia oprogramowania. Zapewnia infrastrukturę do uruchamiania, rozwijania i zarządzania aplikacjami na platformie Windows oraz innych systemach operacyjnych.

Jedną z charakterystycznych cech .NET Framework jest jego wieloplatformowość, co oznacza możliwość pisania kodu raz i uruchamiania go na różnych systemach operacyjnych, takich jak Windows, Linux czy macOS. Taka elastyczność sprawia, że framework ten jest atrakcyjny dla programistów oraz firm poszukujących rozwiązania umożliwiającego tworzenie aplikacji na różnych platformach. W skład .NET Framework wchodzi rozbudowany zestaw bibliotek i narzędzi, które usprawniają proces tworzenia oprogramowania. Obejmuje on między innymi biblioteki do obsługi interfejsu użytkownika, zarządzania pamięcią, komunikacji sieciowej oraz dostępu do danych.

.NET Framework jest zintegrowy innymi technologiami Microsoftu, takimi jak Visual Studio czy platforma chmurowa Azure. Umożliwia to programistom korzystanie z kompleksowych narzędzi do tworzenia, testowania i wdrażania aplikacji, a także skalowania ich w chmurze.

3.3 NestJS

NestJS jest frameworkiem dedykowanym do tworzenia wydajnych, skalowalnych aplikacji serwerowych opartych na Node.js [4]. Jest oparty na progresywnym JavaScriptie, wspiera w pełni TypeScript, pozwalając jednocześnie programistom pisać w czystym JavaScriptie.

Framework NestJS wykorzystuje renomowane platformy HTTP Server, takie jak Express (domyślnie) i opcjonalnie Fastify. Daje to programistom swobodę wyboru między nimi, jednocześnie oferując abstrakcję ponad nimi oraz dostęp do ich interfejsów API.

Filozofia frameworka NestJS opiera się na potrzebie stworzenia spójnej architektury aplikacji, które umożliwiają łatwe testowanie, skalowalność, luźne powiązania oraz łatwe utrzymanie kodu. Inspiracją dla tej architektury były koncepcje stosowane w frameworku Angular.

NestJS jest projektem o otwartym źródle na licencji MIT, rozwijanym od 2017 roku.

Firmy takie jak Adidas, czy Decathlon korzystają z frameworka NestJS [6].

3.4 K6

Grafana k6 to narzędzie do testowania obciążenia aplikacji internetowych oraz wykonywania testów wydajnościowych [2]. Jest to część ekosystemu Grafana, znanej platformy do monitorowania i analizy danych, co zapewnia użytkownikom możliwość integracji testów wydajnościowych z analizą danych i wizualizacją wyników.

Jedną z kluczowych cech narzędzia Grafana k6 jest jego zdolność do symulowania zachowania użytkowników poprzez wysyłanie zapytań HTTP i analizowanie odpowiedzi serwera. Można tworzyć zaawansowane scenariusze testowe, które odwzorowują różne zachowania użytkowników na stronie internetowej, takie jak logowanie, przeglądanie stron, czy też dodawanie produktów do koszyka. Oferuje ono również bogate możliwości konfiguracyjne, które pozwalają dostosować testy do różnych

scenariuszy. Można określić warunki obciążeniowe, definiować progi wydajnościowe oraz zbierać szczegółowe dane diagnostyczne, które pomagają zidentyfikować przyczyny ewentualnych problemów.

Kod źródłowy k6 jest dostępny publicznie, co oznacza, że jest dostępny dla szerokiej społeczności deweloperów i testerów. Dzięki temu można korzystać z bogatej dokumentacji, zgłaszać błędy oraz współpracować nad rozwojem narzędzia w ramach społeczności.

Grafana k6 to wszechstronne i potężne narzędzie do testowania wydajności aplikacji internetowych, które pozwala użytkownikom na symulowanie różnych scenariuszy obciążeniowych oraz monitorowanie wydajności. Dzięki temu deweloperzy i testerzy mogą zapewnić, że ich aplikacje są wydajne i odpowiadają na oczekiwania użytkowników.

3.5 Docker

Docker to platforma wirtualizacyjna, opublikowana przez firmę Docker, Inc. w marcu 2013 roku [5], która umożliwia tworzenie, uruchamianie oraz zarządzanie kontenerami aplikacji [7]. Kontenery są izolowanymi jednostkami oprogramowania, które zawierają wszystkie niezbędne zależności oraz środowisko uruchomieniowe potrzebne do poprawnego działania aplikacji. Docker wykorzystuje technologię konteneryzacji, co pozwala na przenośność aplikacji między różnymi środowiskami oraz zapewnia spójność działania na różnych platformach.

Jedną z kluczowych cech Dockera jest jego szybkość oraz lekkość. Kontenery są uruchamiane na poziomie systemu operacyjnego gospodarza, co eliminuje narzut związany z wirtualizacją tradycyjną. Dzięki temu aplikacje uruchomione w kontenerach Docker są bardzo wydajne i mogą być łatwo skalowane w zależności od obciążenia.

Docker jest również popularnym narzędziem w środowiskach deweloperskich oraz produkcyjnych. Pozwala on programistom tworzyć odizolowane środowiska programistyczne, co ułatwia testowanie oraz debugowanie aplikacji. Jest powszechnie stosowany w procesie wdrażania aplikacji, umożliwiając szybkie i spójne wdrożenie aplikacji na różne serwery oraz platformy chmurowe. Dzięki wsparciu dla narzędzi takich jak Kubernetes czy Docker Swarm, Docker jest używany do budowy i zarządzania dużymi klastrami kontenerów, co umożliwia elastyczne i skalowalne wdrożenia aplikacji w różnych środowiskach produkcyjnych.

Rozdział 4

Metody badawcze

Ważnym aspektem badań naukowych jest efektywne zarządzanie czasem, konieczne do osiągnięcia pożądanych wyników. Często precyzyjność pomiarów nie jest decydująca, lecz wystarczająco dokładna przybliżona wartość. Dlatego też, przed przystąpieniem do dłuższych badań, często przeprowadza się szybkie testy w celu wstępnego oszacowania oczekiwanych rezultatów. Ten proces pozwala na lepsze zrozumienie potencjalnych wyników oraz skuteczne wykorzystanie zasobów badawczych. W rezultacie, efektywność czasowa staje się kluczowym elementem strategii badawczej, prowadząc do bardziej skutecznych i wydajnych działań naukowych.

Przy większych rozwiązaniach stosowana jest horyzontalna architektura pozwalająca rozłożyć większy ruch użytkowników. W badaniach została zastosowana jedna jednostka aplikacji. Należy założyć, że obsługa większej ilości użytkoników może być łatwo uzyskana poprzez zwielokrotnienie uruchomionych instancji. Każdy z badanych frameworków jest przygotowany do obsługi architektury horyzontalnej.

4.1 Badanie pojedyńczego zapytania

Rozpocząłem moje badania od analizy pojedynczego zapytania w wybranych frameworkach. Głównym celem tego eksperymentu było zbadanie czasu potrzebnego do uzyskania danych z bazy danych. Wykorzystałem zbiory danych oznaczone jako FWB_0 oraz FWB_100K. Warto zaznaczyć, że każdy z analizowanych frameworków operował na tych samych zbiorach danych, co umożliwiło porównanie szybkości odpowiedzi na zapytanie między badanymi narzędziami.

Wyniki tego badania pozwalają na początkową weryfikację efektywności poszczególnych frameworków w obsłudze pojedynczych zapytań, co jest kluczowe przy projektowaniu aplikacji. Analiza czasu odpowiedzi na zapytania umożliwia identyfikację potencjalnych obszarów do poprawy oraz wybranie obszarów potrzebujących usprawnienia. Jest to pierwszy krok w celu porównania efektywności narzędzi pomagając w wyborze optymalnej technologii w zależności od konkretnych potrzeb projektowych. Wnioski z tego badania mają istotne znaczenie dla procesu decyzyjnego związanego z wyborem odpowiednich narzędzi i technologii.

4.2 Badania limitu użytkowników

Kolejnym etapem badań było zbadanie granic ilości użytkowników, którymi wybrane narzędzia mogą obsłużyć równolegle. Za kryterium niepowodzenia przyjęto sytuacje, w których narzędzie zwracało błąd lub nie udawało się uzyskać odpowiedzi w określonym czasie. W celu określenia tych granic, przeprowadzono serię testów z różnymi liczbami użytkowników, wykorzystując metodę wyszukiwania binarnego. Warto podkreślić, że uzyskane wyniki są przybliżone, ponieważ mogą nieco się różnić w zależności od warunków testowych. NAnaliza granic jest kluczowa dla oceny skalowalności i wydajności badanych narzędzi w kontekście obsługi większej liczby użytkowników. Otrzymane rezultaty pozwolają na lepsze zrozumienie możliwości i ograniczeń poszczególnych frameworków, co przyczynia się do bardziej świadomego wyboru technologii w projektach wymagających obsługi wielu użytkowników jednocześnie.

Analiza granic ilości użytkowników obsługiwanych równolegle przez badane narzędzia jest istotna z perspektywy projektowania i skalowania systemów, zwłaszcza w kontekście aplikacji o dużej liczbie użytkowników. Poznanie tych granic pozwala na określenie optymalnej konfiguracji środowiska oraz planowanie zasobów potrzebnych do obsługi oczekiwanej liczby użytkowników. Identyfikacja punktów granicznych umożliwia programistom dostosowanie strategii zarządzania obciążeniem oraz wprowadzenie optymalizacji w celu poprawy wydajności systemu. Warto pamiętać, że wyniki tych testów mogą być podatne na zmiany w zależności od czynników zewnętrznych i warunków testowych.

Rozdział 5

Przygotowanie aplikacji

W celu przeprowadzenia badania nad wydajnością oraz porównaniem trzech różnych aplikacji serwujących podobne API, zdecydowano się na implementację trzech aplikacji w różnych technologiach: Django, .NET oraz NestJS. Każda z aplikacji została skonfigurowana do korzystania z bazy danych PostgreSQL. Całe środowisko, a więc aplikacja oraz baza danych, zostało uruchomione w kontenerach Docker w lokalnym środowisku.

Po zakończeniu implementacji każdej z aplikacji, przygotowano kilka scenariuszy testowych, które miały być użyte do oceny wydajności każdej z aplikacji. Do przeprowadzenia testów wydajnościowych wykorzystano narzędzie Grafana k6, które umożliwiło monitorowanie i symulację obciążenia na aplikacji. Napisane zostały skrypty które uruchamiją test k6, a następnie analizują wyniki.

5.1 Zbiory danych

W celu przeprowadzenia badania konieczne było przygotowanie odpowiednich zbiorów danych, które miały posłużyć do symulacji różnych scenariuszy. W tym kontekście przygotowano dwa zbiory danych, aby umożliwić różnorodne analizy:

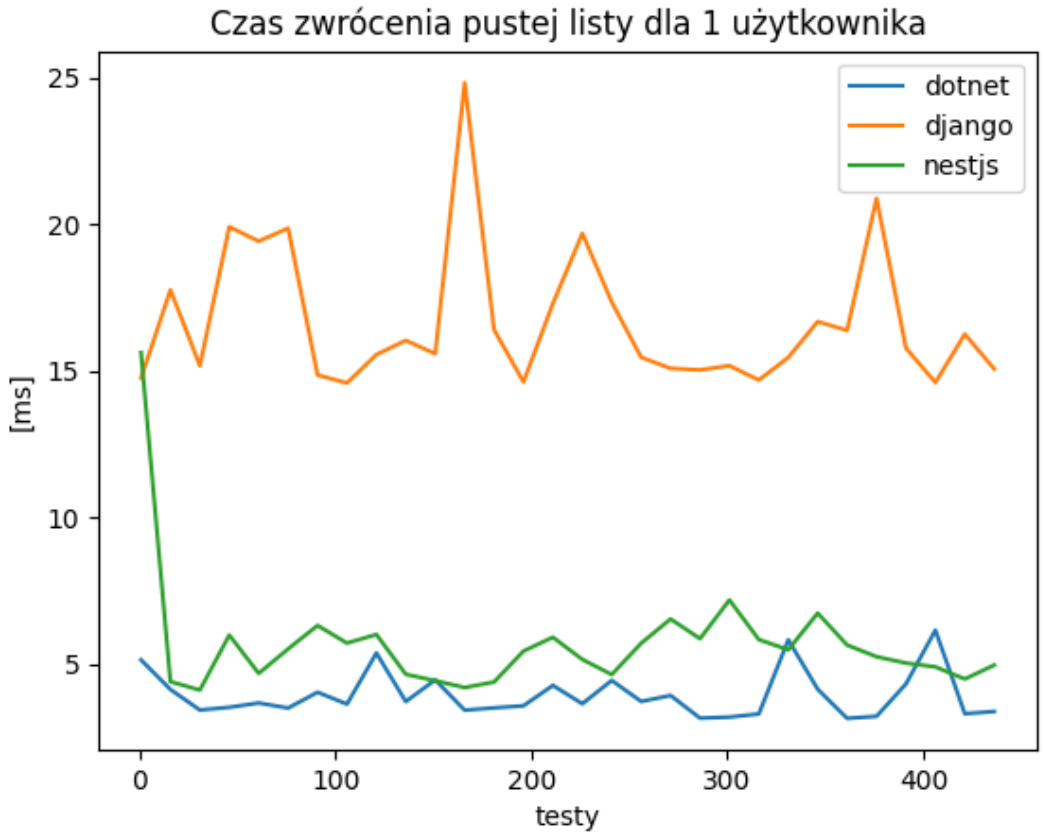
- **FWB_0** - Jest to zbiór pusty, pozbawiony jakichkolwiek elementów. Brak danych w tym zbiorze ma posłużyć do sprawdzenia zachowania systemu w sytuacji, gdy nie ma żadnych rekordów do przetworzenia.
- **FWB_100K** - Ten zbiór składa się z 100 000 elementów. Każdy element tego zbioru reprezentuje pojedynczy rekord w bazie danych i zawiera unikalne identyfikatory (numery) oraz nazwy (tekstowe). Zbiór ten został przygotowany w celu przetestowania wydajności systemu oraz jego reakcji na duże ilości danych.

Jeden element to rekord w bazie danych zawierający ID (number) oraz nazwę (tekst). Przygotowanie tych zbiorów danych stanowiło niezbędny krok przed przystąpieniem do właściwej analizy i symulacji różnych scenariuszy w badaniu. Dzięki tym zbiorom możliwe było zbadanie zachowania systemu w różnych warunkach oraz przeprowadzenie odpowiednich wniosków na podstawie uzyskanych wyników.

Rozdział 6

Wyniki i dyskusja

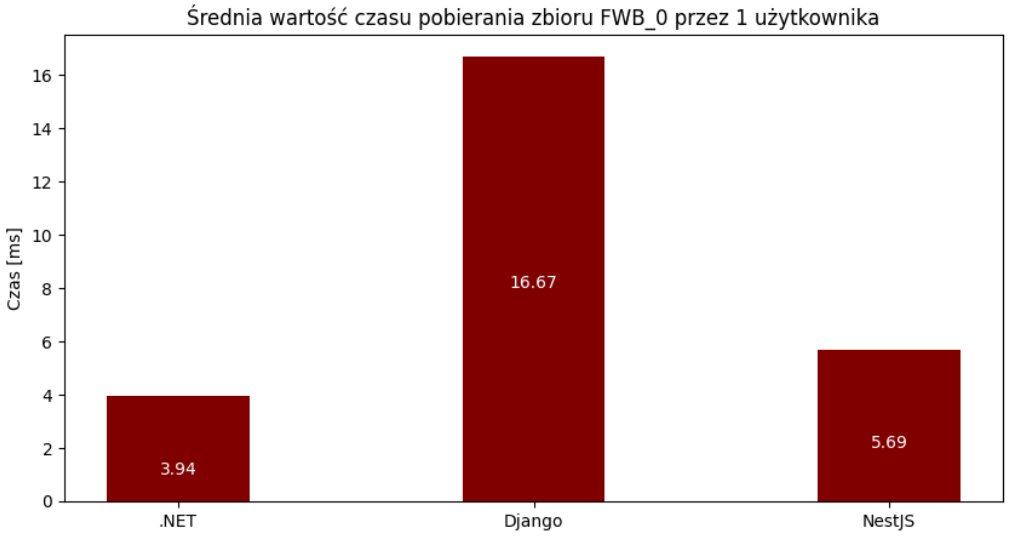
6.1 Pojedyńcze zapytania



Rysunek 1. Czas zwrócenia pustej listy dla 1 użytkownika

Zmierzone wartości otrzymane podczas badania zostały zaprezentowane na rysunku 1. Prezentuje on czas zwrócenia wartości przez framwork w mierzonym oknie czasowym. Standardowe odchylenie

dla czas zmierzononego dla frameworka .NET wyniosło 0,78 ms. Było to najmniejsze wahanie na tle innych badanych narzędzi. Dla Django standardowe odchylenie to 2,40 ms. Trochę mniejsze odchylenie uzyskał NestJS uzyskując wartość 2,03 ms. Widoczne jest większe odchylenie podczas pierwszego zapytania. Późniejsze zapytania dostają odpowiedź zdecydowanie szybciej.



Rysunek 2. Średni czas zwrócenia pustej listy dla 1 użytkownika

Średni czas zwrócenia odpowidzi podczas tego badania został zaprezentowany na rysunku 2. Zdycydowanie najdłuższy odpowiedzi przpadł Django. Około 3 razy mniejszy czas uzyskał NestJS. Wygranym tego zestawienia został .NET uzyskując najszybszy średni czas odpowiedzi.

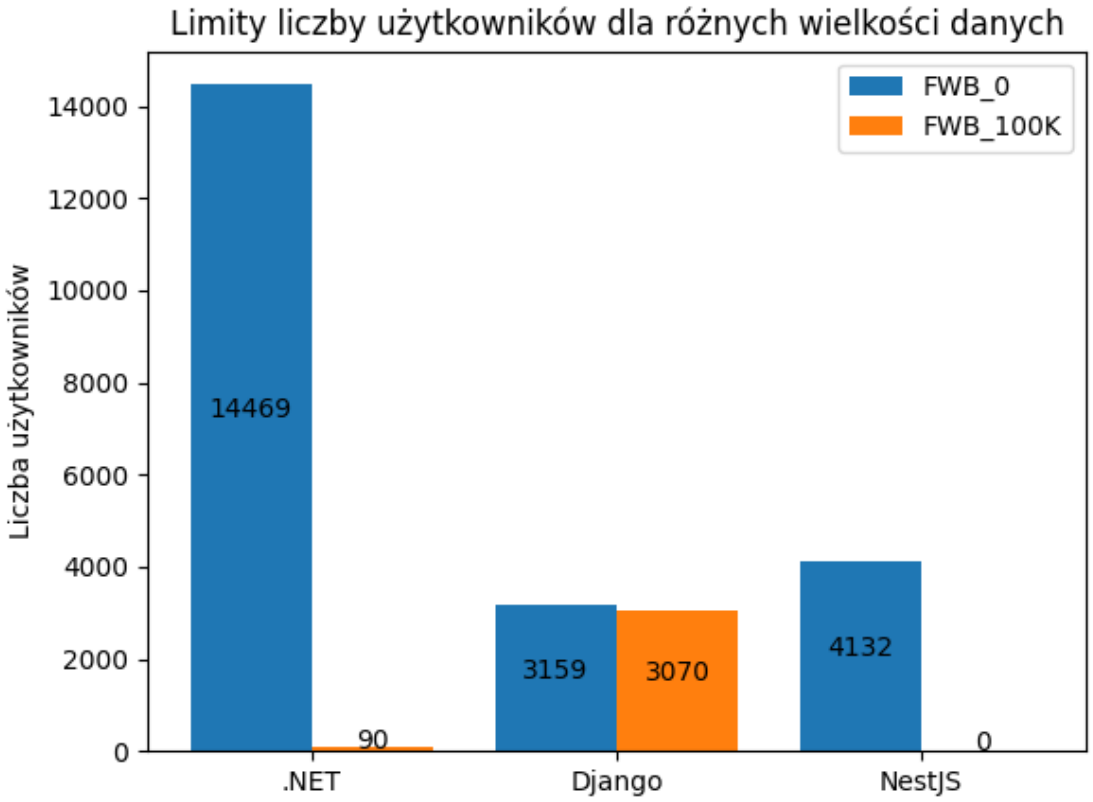
6.2 Limity równoległych zapytań

Wyniki badania limitów zostały zaprezentowane na rysunku 3.

Django potrafi obsługiwać podobną ilość użytkowników dla obu zbiorów. W przypadku zbioru danych FWB_100K, narzędzie zanotowało spadek o niecałe 3%.

.NET jest zdecydowanym liderem liczby użytkowników dla pustego zbioru. Widać u niego jednak znaczny spadek liczby obsługiwanych użytkowników wraz ze wzrostem ilości przesyłanych danych.

NestJS które przy zbiorze pustym potrafi obsłużyć nieco więcej użytkowników od Django, dla zbioru FWB_100K całkowicie przestało odpowiadać - co oznacza, że nie radzi sobie z taką ilością danych.



Rysunek 3. Limit liczby użytkowników dla zwracajnych różnych wielkości danych dla jednej instacji frameworka

Rozdział 7

Podsumowanie i wnioski

Badanie pojedyńczego wskazało że Django jest zdecydowanie wolniejszym framworkiem od pozostałych dwóch. Badanie limitów wskazało istotną słabość NestJS polegającą na problemie obsługi dużych ilości danych. .NET wykazał się istotnm spadkiem ilości obsługiwanych uzytkowników. Dla Django duże zbiory danych niewiele wpłynęły na liczbę obsugiwanych użytkowników.

Wybór narzędzia jest uzależniony od konkretnego zastosowania. Bardzo często istotny jest czas pojedyńczego zapytania, ponieważ wpływa mocno na doświadczenie użytkownika z korzystania z aplikacji. Uznając to za istotny czynnik należy odrzucić framework Django jak prezentujący się jako najwolniejszy na tle innych badanych rozwiązań.

Rzadko wymagane jest przesyłanie dużych ilości danych przez API frameworka. Należy zaznaczyć, że do przesyłania plików, a więc potencjalnie więszych danch są osobne rozwiązania, kóre nie zostały zastosowane podczas badania. Zazwyczaj takie dane są przesyłane porcjami. .NET wykazał się lepszym rezultatem w obu badania w porównaniu z NestJS.

Istotnym czynnikiem decydującym o wyborze framworka jest doświadczenie oraz łatwość pisania. Są to czynniki indywidualne które nie podlegały dogłębnej analizie podczas tego badania. Jest to doskonały materiał na rozszerzenie niniejszej pracy. Ostatecznie wybór odpowidniego frameworka ma pomóc z sukcesem zrealizować projekt biznesowy.

Bibliografia

[1] *.NET | Build. Test. Deploy. — dotnet.microsoft.com*, https://dotnet.microsoft.com/, [Accessed 17-03-2024].

[2] *Browser Module Documentation — k6.io*, https://k6.io/docs/using-k6-browser/overview/, [Accessed 17-03-2024].

[3] *Django overview — djangoproject.com*, https://www.djangoproject.com/start/overview/, [Accessed 17-03-2024].

[4] *Documentation | NestJS - A progressive Node.js framework — docs.nestjs.com*, https://docs.nestjs.com/, [Accessed 17-03-2024].

[5] *dotCloud - About — web.archive.org*, https://web.archive.org/web/20140702231323/https://www.dotcloud.com/about.html, [Accessed 17-03-2024].

[6] *Here's How to Hire NestJS Developers — revelo.com*, https://www.revelo.com/blog/hire-nestjs-developers, [Accessed 17-03-2024].

[7] *Home — docs.docker.com*, https://docs.docker.com/, [Accessed 17-03-2024].

Wykaz skrótów i symboli

Spis rysunków

Spis tablic

Spis załączników

Załącznik 1
Dowód próżni doskonałej

Fusce mauris. Vestibulum luctus nibh at lectus. Sed bibendum, nulla a faucibus semper, leo velit ultricies tellus, ac venenatis arcu wisi vel nisl. Vestibulum diam. Aliquam pellentesque, augue quis sagittis posuere, turpis lacus congue quam, in hendrerit risus eros eget felis. Maecenas eget erat in sapien mattis porttitor. Vestibulum porttitor. Nulla facilisi. Sed a turpis eu lacus commodo facilisis. Morbi fringilla, wisi in dignissim interdum, justo lectus sagittis dui, et vehicula libero dui cursus dui. Mauris tempor ligula sed lacus. Duis cursus enim ut augue. Cras ac magna. Cras nulla. Nulla egestas. Curabitur a leo. Quisque egestas wisi eget nunc. Nam feugiat lacus vel est. Curabitur consectetuer.

Załącznik 2
Dowód zera bezwzględnego

Suspendisse vel felis. Ut lorem lorem, interdum eu, tincidunt sit amet, laoreet vitae, arcu. Aenean faucibus pede eu ante. Praesent enim elit, rutrum at, molestie non, nonummy vel, nisl. Ut lectus eros, malesuada sit amet, fermentum eu, sodales cursus, magna. Donec eu purus. Quisque vehicula, urna sed ultricies auctor, pede lorem egestas dui, et convallis elit erat sed nulla. Donec luctus. Curabitur et nunc. Aliquam dolor odio, commodo pretium, ultricies non, pharetra in, velit. Integer arcu est, nonummy in, fermentum faucibus, egestas vel, odio.

Załącznik 3

Dowód czasu zatrzymanego

Sed commodo posuere pede. Mauris ut est. Ut quis purus. Sed ac odio. Sed vehicula hendrerit sem. Duis non odio. Morbi ut dui. Sed accumsan risus eget odio. In hac habitasse platea dictumst. Pellentesque non elit. Fusce sed justo eu urna porta tincidunt. Mauris felis odio, sollicitudin sed, volutpat a, ornare ac, erat. Morbi quis dolor. Donec pellentesque, erat ac sagittis semper, nunc dui lobortis purus, quis congue purus metus ultricies tellus. Proin et quam. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Praesent sapien turpis, fermentum vel, eleifend faucibus, vehicula eu, lacus.

Załącznik 4
Dowód nieskończoności urojonej

Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Donec odio elit, dictum in, hendrerit sit amet, egestas sed, leo. Praesent feugiat sapien aliquet odio. Integer vitae justo. Aliquam vestibulum fringilla lorem. Sed neque lectus, consectetuer at, consectetuer sed, eleifend ac, lectus. Nulla facilisi. Pellentesque eget lectus. Proin eu metus. Sed porttitor. In hac habitasse platea dictumst. Suspendisse eu lectus. Ut mi mi, lacinia sit amet, placerat et, mollis vitae, dui. Sed ante tellus, tristique ut, iaculis eu, malesuada ac, dui. Mauris nibh leo, facilisis non, adipiscing quis, ultrices a, dui.